Endeca® Platform Services

Data Foundry Expression Reference

Version 6.1.1• December 2011

Contents

Copyright and disclaimer

Preface

Endeca® InFront enables businesses to deliver targeted experiences for any customer, every time, in any channel. Utilizing all underlying product data and content, businesses are able to influence customer behavior regardless of where or how customers choose to engage — online, in-store, or on-the-go. And with integrated analytics and agile business-user tools, InFront solutions help businesses adapt to changing market needs, influence customer behavior across channels, and dynamically manage a relevant and targeted experience for every customer, every time.

InFront Workbench with Experience Manager provides a single, flexible platform to create, deliver, and manage content-rich, multichannel customer experiences. Experience Manager allows non-technical users to control how, where, when, and what type of content is presented in response to any search, category selection, or facet refinement.

At the core of InFront is the Endeca MDEX Engine,™ a hybrid search-analytical database specifically designed for high-performance exploration and discovery. InFront Integrator provides a set of extensible mechanisms to bring both structured data and unstructured content into the MDEX Engine from a variety of source systems. InFront Assembler dynamically assembles content from any resource and seamlessly combines it with results from the MDEX Engine.

These components — along with additional modules for SEO, Social, and Mobile channel support — make up the core of Endeca InFront, a customer experience management platform focused on delivering the most relevant, targeted, and optimized experience for every customer, at every step, across all customer touch points.

About this guide

This reference describes the Data Foundry expressions available for use in a record manipulator component in Developer Studio.

Who should use this guide

This reference is intended for developers who are building Data Foundry pipelines using Endeca Developer Studio.

Conventions used in this guide

This guide uses the following typographical conventions:

Code examples, inline references to code elements, file names, and user input are set in `monospace` font. In the case of long lines of code, or when inline monospace text occurs at the end of a line, the following symbol is used to show that the content continues on to the next line: ¬

When copying and pasting such examples, ensure that any occurrences of the symbol and the corresponding line break are deleted and any remaining space is closed up.

Contacting Endeca Customer Support

The Endeca Support Center provides registered users with important information regarding Endeca software, implementation questions, product and solution help, training and professional services consultation as well as overall news and updates from Endeca.

You can contact Endeca Standard Customer Support through the Support section of the Endeca Developer Network (EDeN) at *http://eden.endeca.com*.

Chapter 1

About Data Foundry expressions

XML syntax for EXPRESSION elements

EXPRESSION

An EXPRESSION element instructs Forge about how to modify records. An expression consists of an EXPRESSION element with TYPE and NAME attributes.

Expressions may contain EXPRNODE sub-elements, which have NAME and VALUE attributes, to supply additional configuration information. Expressions may also contain other expressions; the contained expressions may provide values used by the containing expression, or the containing expression may provide control over which of the contained expressions are evaluated.

Endeca recommends that you perform record manipulation with a Perl manipulator rather than use a record manipulator and EXPRESSION elements. However, if you need to access and modify dimension sources (such as a dimension adapter or dimension server), you should still use expressions such as DVAL PERL. The Perl manipulator does not access dimension sources. For more information about using a Perl manipulator, see the *Endeca Developer Studio Help*.

DTD

```
<!ELEMENT EXPRESSION

    ( COMMENT?
    , EXPRBODY?
    , (EXPRNODE | EXPRESSION)*
    )
>
<!ATTLIST EXPRESSION
    TYPE ( PROPERTY
         | DVAL
         | INTEGER
         | STRING
         | STREAM
         | VOID
         | FLOAT)            #REQUIRED
    NAME        CDATA        #REQUIRED
    LABEL       CDATA        #IMPLIED
    URL         CDATA        #IMPLIED
>
```

Attributes

The following points describe the EXPRESSION element's attributes.

- TYPE - Describes the return value for the expression. For example, a FLOAT expression returns a floating point number. The valid values for TYPE are as follows: PROPERTY, DVAL, INTEGER, STRING, VOID, and FLOAT.

   **Note:** STREAM is used internally by the MDEX Engine.

- NAME - Describes the operation being performed. Expressions are typically referred to by the combination of their TYPE and NAME values (for example DVAL CONST). This combination helps to distinguish cases where there are several expressions with different TYPE values but the same NAME value (for example, DVAL CONST, FLOAT CONST, and INTEGER CONST).
- URL - Used in PERL expressions. Specifies the URL (file) from which an expression can read Perl code. The code can be up to 65534 characters long.

Sub-elements

The following table provides a brief overview of the EXPRESSION sub-elements.

Sub-element	Description
COMMENT	Associates a comment with a parent element and preserves the comment when the file is rewritten. This element provides an alternative to using inline XML comments of the form .
EXPRBODY	Contains Perl code that manipulates records.
EXPRNODE	Provides a generic way of sending a variety of information to an EXPRESSION.
EXPRESSION	Instructs Forge about how to modify records.

Example

This example shows a mathematical expression that adds two constant values (5 and 6).

```
<EXPRESSION TYPE="INTEGER" NAME="MATH">
   <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
   <EXPRNODE NAME="OPERATOR" VALUE="ADD"/
   <EXPRESSION TYPE="INTEGER" NAME="CONST">
      <EXPRNODE NAME="VALUE" VALUE="5"/>
   </EXPRESSION>
   <EXPRESSION TYPE="INTEGER" NAME="CONST">
      <EXPRNODE NAME="VALUE" VALUE="6"/>
   </EXPRESSION>
</EXPRESSION>
```

This example adds a dimension value ID to the current record. The dimension value ID is determined by the mapping between the value of the P_Score property in the source data and the dimension values contained in the dimension with ID equal to 9.

```
<EXPRESSION NAME="ADD_DVAL" TYPE="VOID">
   <EXPRESSION NAME="MATCH" TYPE="DVAL">
      <EXPRNODE NAME="DIMENSION_ID" VALUE="9"/>
      <EXPRESSION NAME="IDENTITY" TYPE="PROPERTY">
         <EXPRNODE NAME="PROP_NAME" VALUE="P_Score"/>
```

```
      </EXPRESSION>
   </EXPRESSION>
</EXPRESSION>
```

EXPRNODE

An EXPRNODE element provides a generic way of sending a variety of information to an EXPRESSION. The information could be descriptions, data types, constant values (parameters), and so on. Comparatively speaking, this information is similar to a parameter for a function.

DTD

```
<!ELEMENT EXPRNODE (EXPRNODE*)>
<!ATTLIST EXPRNODE
    NAME    CDATA   #REQUIRED
    VALUE   CDATA   #IMPLIED
>
```

Attributes

The following points describe the EXPRNODE element's attributes.

- NAME - Describes the EXPRNODE element. Because EXPRNODE can be so broadly used to modify an expression, the NAME attribute varies in relation to the expression that it modifies. For example, NAME can specify a variety of values such as TYPE, NAME, OPERATOR, AUTO_GEN, OPERATION, and so on. See an expression's help topic for details about how the NAME attribute of an expression node modifies an expression.
- VALUE - Provides a value that corresponds to the NAME attribute. Because EXPRNODE can be so broadly used, the VALUE attribute can specify a variety of values. For example, INTEGER may correspond to TYPE; SUM may correspond to NAME; ADD may correspond to OPERATOR and so on. See an expression's help topic for details about how the VALUE attribute of an expression node modifies an expression.

Sub-elements

The EXPRNODE element can contain additional EXPRNODE elements as sub-elements.

Example

This example shows an expression adding a dimension value ID to the current record. The second and third expressions use EXPRNODE to provide name and value information for the parent EXPRES¬ SION.

```
<EXPRESSION NAME="ADD_DVAL" TYPE="VOID">
   <EXPRESSION NAME="MATCH" TYPE="DVAL">
      <EXPRNODE NAME="DIMENSION_ID" VALUE="9"/>
      <EXPRESSION NAME="IDENTITY" TYPE="PROPERTY">
         <EXPRNODE NAME="PROP_NAME" VALUE="P_Score"/>
      </EXPRESSION>
   </EXPRESSION>
</EXPRESSION>
```

EXPRBODY

An EXPRBODY element contains Perl code that manipulates data. Perl code is often useful if a data manipulation task is too complicated to perform using the Data Foundry expressions. EXPRBODY is a child of EXPRESSION.

Endeca recommends that you perform record manipulation with a Perl manipulator rather than with a record manipulator that uses EXPRESSION elements. However, if you need to access and modify dimension sources (e.g., a dimension adapter or dimension server), you should still use expressions such as DVAL PERL. The Perl manipulator does not access dimension sources. For more information about using a Perl manipulator, see the *Endeca Developer Studio Help*.

DTD

```
<!ELEMENT EXPRBODY (#PCDATA)>
```

Attributes

The EXPRBODY element has no attributes.

Sub-elements

The EXPRBODY element has no sub-elements.

Example

This example shows the outline of an expression using Perl.

```
<EXPRESSION TYPE=”VOID” NAME=”PERL”>
   <EXPRBODY>
   ...Perl code here...
   </EXPRBODY>
</EXPRESSION>
```

DVAL expressions

DVAL CONST

DVAL expressions return dimension values. DVAL CONST expressions return the dimension value (or set of dimension values), specified in DIMENSION_ID/DVAL_ID expression node pairs.

For example, using a DVAL CONST expression within a VOID ADD_DVAL expression instructs Forge to add the dimension value to each record processed. The DIMENSION_ID and DVAL_ID expression nodes may be repeated in pairs to create sets of dimension values. Forge then adds the set of dimension values to each record processed.

See the EXPRESSION element for DTD and attribute information.

Example

This example adds the dimension value to the record being processed.

```
<EXPRESSION TYPE="VOID" NAME="ADD_DVAL">
   <EXPRESSION TYPE="DVAL" NAME="CONST">
      <EXPRNODE NAME="DIMENSION_ID" VALUE="2090"/>
```

```
      <EXPRNODE NAME="DVAL_ID" VALUE="2091"/>
   </EXPRESSION>
</EXPRESSION>
```

DVAL MATCH

DVAL expressions return dimension values. Although the DVAL MATCH expression is not deprecated, we strongly recommend that you use the PROP_MAPPER element to perform property and dimension value mapping operations.

The DVAL MATCH expression is used to perform text matching within one or more dimensions. The dimensions are specified in DIMENSION_ID EXPRNODE elements. Specify the values to match in one or more PROPERTY expressions. The expression returns a set of matching dimension values. This is useful within a VOID ADD_DVAL expression that contains PROPERTY IDENTITY sub-expressions, to assign the dimension values matching each record's property values to the record.

You can adjust the behavior of the DVAL MATCH expression by nesting EXPRNODE elements within the DVAL MATCH expression. The nested expression node has a NAME attribute that specifies how to adjust the behavior.

The following list provides the supported values of the NAME attribute and describes their effects on the behavior of DVAL MATCH.

- AUTO_GEN - Causes new dimension values to be generated automatically to match property values that do not already match dimension values within the dimension. The VALUE attribute for AUTO_GEN may be either TRUE or FALSE. The default value is TRUE.
- DEFAULT_SIFT_HIER_DEPTH - Builds a auto-generated sift hierarchy to the depth specified in the VALUE attribute. For example, <EXPRNODE NAME="DEFAULT_SIFT_HIER_DEPTH" VAL¬ UE="2"/> builds to a depth of two. See "Working with Large Dimension Hierarchies" in the *Endeca Developer Studio Help* for details.
- LOG - Identifies the LOG to which MATCH errors should be written. To avoid duplicate error messages, this is normally a unique log. See the LOG element's TYPE attribute to specify unique error message logging.
- MATCH_EMPTY_PROPS - Specifies that Forge create an empty dimension value for any empty property value Forge finds during matching. This empty property to dimension value matching occurs when you use AUTO_GEN with MATCH_EMPTY_PROPS set to TRUE. If this expression node is set to FALSE or omitted, Forge ignores empty properties.
- MUST_MATCH - Specifies that every property value must match a dimension value within the specified dimensions. If a property value does not have a match, an error is written out to the log named in the LOG expression node. Note that MUST_MATCH should not be used in combination with AUTO_GEN, because all property values match if AUTO_GEN is specified.
- REMOVE_PROP - Removes the properties used for the MATCH from the current record after the match is performed. This is useful if the dimension is being indexed, but not the source property.
- SIFT_MATCH - SIFT_MATCH is a special variation of AUTO_GEN. See the *Endeca Basic Development Guide* for details.
- SYN_MATCH - Uses the name of the first property value that matches a dimension value as the synonym name for any property values that do not match to a dimension value. The Forge matching process goes as follows: First, AUTO_GEN must set be FALSE. During matching, if more than one property value matches a dimension value, Forge logs an error. If a property value does not match a dimension value, Forge creates a new dimension value using the property value as its name. All the non-matching property values are turned into synonyms of this dimension value.

See the EXPRESSION element for DTD and attribute information.

Example

This example assigns the dimension value 2090 to records with the property name FORMAT.

```
<EXPRESSION TYPE="VOID" NAME="ADD_DVAL">
   <EXPRESSION TYPE="DVAL" NAME="MATCH">
      <EXPRNODE NAME="DIMENSION_ID" VALUE="2090"/>
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="FORMAT"/>
         <EXPRNODE NAME="LOG" VALUE="salog"/>
      </EXPRESSION>
   </EXPRESSION>
</EXPRESSION>
```

DVAL PERL

DVAL expressions return dimension values. The DVAL PERL expression uses Perl to manipulate the data.

The Perl code is contained in an EXPRBODY element. The Zinc Perl module provides classes and methods for gaining access to and manipulating the current record. Objects accessed from Perl are copies of the current data; changing the Perl objects has no effect on the current data until a function is called to explicitly copy the Perl objects back.

See the EXPRESSION element for DTD and attribute information.

Example

This example assigns the dimension value 2090 to records with the property name FORMAT.

```
<EXPRESSION TYPE="DVAL" NAME="PERL">
   <EXPRBODY>
   ...your Perl code here. The VOID PERL topic has an example...
   </EXPRBODY>
</EXPRESSION>
```

FLOAT expressions

FLOAT CONST

FLOAT expressions return floating point (fractional) numbers. The FLOAT CONST expression returns the same floating-point number, specified in the VALUE attribute of an EXPRNODE sub-element.

For example, using a FLOAT CONST expression within a VOID CREATE expression adds the new property whose value is the constant specified to each record processed.

See the EXPRESSION element for DTD and attribute information.

Example

This example deletes discount properties if they are less than 20%. The FLOAT CONST expression defines the value for the percentage.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="MATH">
      <EXPRNODE NAME="TYPE" VALUE="FLOAT"/>
```

```
      <EXPRNODE NAME="OPERATOR" VALUE="LT"/>
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="DISCOUNT"/>
      </EXPRESSION>
      <EXPRESSION TYPE="FLOAT" NAME="CONST">
         <EXPRNODE NAME="VALUE" VALUE="20.00"/>
      </EXPRESSION>
   </EXPRESSION>
   <EXPRESSION TYPE="VOID" NAME="REMOVE">
      <EXPRNODE NAME="PROP_NAME" VALUE="DISCOUNT"/>
   </EXPRESSION>
</EXPRESSION>
```

FLOAT MATH

FLOAT expressions return floating point (fractional) numbers. The FLOAT MATH expression performs a floating-point arithmetic operation on two values.

The operation to be performed is supplied in the VALUE attribute of an EXPRNODE element. The values to be operated on are supplied in two sub-expressions.

The possible operations that may be specified in the VALUE attribute are as follows:

- ADD
- SUBTRACT - expression 1 minus expression 2.
- MULTIPLY
- DIVIDE - expression 1 divided by expression 2.
- POWER - expression 1 raised to the power of expression 2.
- PERCENT - the percentage expression 1 is of expression 2 (100 * (expression 1 / expression 2)).

The sub-expressions can be PROPERTY, STRING, INTEGER, or FLOAT expressions (use PROPERTY expressions to retrieve values from the current record). The values returned by the sub-expressions are converted to floating-point numbers prior to performing the parent operation.

See the EXPRESSION element for DTD and attribute information.

Example

This example subtracts to constants and returns a floating point result.

```
<EXPRESSION TYPE="FLOAT" NAME="MATH">
   <EXPRNODE NAME="OPERATOR" VALUE="SUBTRACT"/>
   <EXPRESSION TYPE="FLOAT" NAME="CONST">
      <EXPRNODE NAME="VALUE" VALUE="9.25"/>
   </EXPRESSION>
   <EXPRESSION TYPE="FLOAT" NAME="CONST">
      <EXPRNODE NAME="VALUE" VALUE="11.75"/>
   </EXPRESSION>
</EXPRESSION>
```

INTEGER expressions

INTEGER AND

INTEGER expressions return integers (whole numbers). INTEGER expressions can be used to combine expressions, do arithmetic, and test conditions for conditional evaluation. The INTEGER AND expression evaluates one or more INTEGER expressions, returning 1 if all the sub-expressions return non-zero values, and otherwise returning 0.

The evaluation of sub-expressions stops as soon as one returns 0. INTEGER AND is the equivalent of the "&&" operator in Perl and C. Used in conjunction with an IF expression, an AND expression can check for more than one condition. The syntax requires one or more nested INTEGER expressions.

See the EXPRESSION element for DTD and attribute information.

Example

This example uses an INTEGER AND expression to evaluate whether the records processed have a subject and sales rank property. If the records do not such properties, the REMOVE_RECORD deletes them.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="AND">
      <EXPRESSION TYPE="INTEGER" NAME="MATH">
         <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
         <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
         <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
            <EXPRNODE NAME="PROP_NAME" VALUE="CATEGORY_ID"/>
         </EXPRESSION>
         <EXPRESSION TYPE="INTEGER" NAME="CONST">
            <EXPRNODE NAME="VALUE" VALUE="0"/>
         </EXPRESSION>
      </EXPRESSION>
      <EXPRESSION TYPE="INTEGER" NAME="MATH">
         <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
         <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
         <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
            <EXPRNODE NAME="PROP_NAME" VALUE="SALESRANK"/>
         </EXPRESSION>
         <EXPRESSION TYPE="INTEGER" NAME="CONST">
            <EXPRNODE NAME="VALUE" VALUE="0"/>
         </EXPRESSION>
      </EXPRESSION>

   </EXPRESSION>
   <EXPRESSION TYPE="VOID" NAME="REMOVE_RECORD"/>
</EXPRESSION>
```

INTEGER CONST

INTEGER expressions return integers (whole numbers). INTEGER expressions can be used to combine expressions, do arithmetic, and test conditions for conditional evaluation. The INTEGER CONST expression returns the same integer constant, specified in the VALUE attribute of an EXPRNODE element.

For example, using an INTEGER CONST expression within a VOID CREATE expression adds a new property, whose value is the specified constant to each record processed.

See the EXPRESSION element for DTD and attribute information.

Example

As part of an INTEGER AND expression, this example uses an INTEGER CONST sub-expression to test whether the PROP_NAME value equals the constant value.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="AND">
      <EXPRESSION TYPE="INTEGER" NAME="MATH">
         <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
         <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
         <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
            <EXPRNODE NAME="PROP_NAME" VALUE="CATEGORY_ID"/>
         </EXPRESSION>
         <EXPRESSION TYPE="INTEGER" NAME="CONST">
            <EXPRNODE NAME="VALUE" VALUE="0"/>
         </EXPRESSION>
      </EXPRESSION>
      <EXPRESSION TYPE="INTEGER" NAME="MATH">
         <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
         <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
         <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
            <EXPRNODE NAME="PROP_NAME" VALUE="SALESRANK"/>
         </EXPRESSION>
         <EXPRESSION TYPE="INTEGER" NAME="CONST">
            <EXPRNODE NAME="VALUE" VALUE="0"/>
         </EXPRESSION>
      </EXPRESSION>
   </EXPRESSION>
   <EXPRESSION TYPE="VOID" NAME="REMOVE_RECORD"/>
</EXPRESSION>
```

INTEGER MATH

INTEGER expressions return integers (whole numbers). INTEGER expressions can be used to combine expressions, do arithmetic, and test conditions for conditional evaluation. The INTEGER MATH expression can perform a variety of operations on two values, including arithmetic, Boolean tests, and string comparison.

Although the returned value is always an integer, the operation itself can be performed using a variety of data types. A TYPE expression node tells the MATH expression what type to convert its sub-expressions into prior to performing the operation. The operation to be performed is supplied in an OPERATOR expression node; the values to be operated on are supplied in two sub-expressions.

In the TYPE expression node, the VALUE attribute has the following supported values:

- STRING
- INTEGER
- FLOAT

The following OPERATOR expression node require that the TYPE attribute of their sub-expressions have a value of either INTEGER or FLOAT:

- ADD
- SUBTRACT - expression 1 minus expression 2.
- MULTIPLY
- DIVIDE - expression 1 divided by expression 2.

- POWER - expression 1 raised to the power of expression 2.
- PERCENT - the percentage expression 1 is of expression 2 (100 * (expression 1 / expression 2)).
- MOD - the remainder of expression 1 divided by expression 2.

The following OPERATOR expression nodes require that the TYPE attribute of their two sub-expressions have a value of INTEGER, FLOAT, or STRING.

- EQUAL - returns 1 if expression 1 and expression 2 are equal, 0 otherwise.
- NE - returns 1 if expression 1 and expression 2 are not equal, 0 otherwise.
- GT - returns 1 if expression 1 is greater than expression 2, 0 otherwise.
- GTE - returns 1 if expression 1 is greater than or equal to expression 2, 0 otherwise.
- LT - returns 1 if expression 1 is less than expression 2, 0 otherwise.
- LTE - returns 1 if expression 1 is less than or equal to expression 2, 0 otherwise.
- CMP - returns 1 if expression 1 is greater than expression 2, 0 if the expressions are equal, –1 if expression 1 is less than expression 2.

The following OPERATOR expression nodes require that the TYPE attribute of their two sub-expressions have a value of STRING:

- CMP_SUBSTR - returns 1 if expression 1 contains expression 2 as a sub-string, 0 otherwise.
- CMP_START - returns 1 if expression 1 starts with expression 2, 0 otherwise.
- CMP_END - returns 1 if expression 1 ends with expression 2, 0 otherwise.

In the OPERATOR expression nodes, the following two values of the VALUE attribute have slightly different behavior for STRING than for INTEGER:

- AND - used with STRING, returns 1 if neither expression 1 nor expression 2 is empty, 0 otherwise. Used with INTEGER, returns 1 if neither expression 1 nor expression 2 is not equal to 0.
- OR - used with STRING, returns 1 if either expression 1 or expression 2 is not empty, 0 otherwise. Used with INTEGER, returns 1 if either expression 1 or expression 2 does not equal 0. See Example 2 for the use of this operator.

See the EXPRESSION element for DTD and attribute information.

Example 1

As part of an INTEGER AND expression, this example uses two INTEGER MATH sub-expressions to test whether the PROP_NAME value equals the constant value.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="AND">
      <EXPRESSION TYPE="INTEGER" NAME="MATH">
         <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
         <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
         <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
            <EXPRNODE NAME="PROP_NAME" VALUE="CATEGORY_ID"/>
         </EXPRESSION>
         <EXPRESSION TYPE="INTEGER" NAME="CONST">
            <EXPRNODE NAME="VALUE" VALUE="0"/>
         </EXPRESSION>
      </EXPRESSION>
      <EXPRESSION TYPE="INTEGER" NAME="MATH">
         <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
         <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
         <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
            <EXPRNODE NAME="PROP_NAME" VALUE="SALESRANK"/>
         </EXPRESSION>
         <EXPRESSION TYPE="INTEGER" NAME="CONST">
            <EXPRNODE NAME="VALUE" VALUE="0"/>
```

```
        </EXPRESSION>
      </EXPRESSION>
   </EXPRESSION>
   <EXPRESSION TYPE="VOID" NAME="REMOVE_RECORD"/>
</EXPRESSION>
```

Example 2

This example illustrates the use of an `OR` in the `OPERATOR` expression node. The example reads: If Category equals "A" or Category equals "B", then create a new instance of the property "ABCompanies" with the value from the Company property. The syntax implicitly selects the first value of a given property if the property is multi-assigned. "IDENTITY" gets the actual value of the property, while "CONST" is a literal.

```
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
  <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
    <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
    <EXPRNODE NAME="OPERATOR" VALUE="OR"/>
    <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
        <EXPRNODE NAME="PROP_NAME" VALUE="Category"/>
      </EXPRESSION>
      <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
        <EXPRNODE NAME="VALUE" VALUE="A"/>
      </EXPRESSION>
    </EXPRESSION>
    <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
        <EXPRNODE NAME="PROP_NAME" VALUE="Category"/>
      </EXPRESSION>
      <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
        <EXPRNODE NAME="VALUE" VALUE="B"/>
      </EXPRESSION>
    </EXPRESSION>
  </EXPRESSION>
  <EXPRESSION LABEL="" NAME="CREATE" TYPE="VOID" URL="">
    <EXPRNODE NAME="PROP_NAME" VALUE="ABCompanies"/>
    <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
      <EXPRNODE NAME="PROP_NAME" VALUE="Company"/>
    </EXPRESSION>
  </EXPRESSION>
</EXPRESSION>
```

INTEGER PERL

`INTEGER` expressions return integers (whole numbers). `INTEGER` expressions can be used to combine expressions, do arithmetic, and test conditions for conditional evaluation. The `INTEGER PERL` expression uses Perl to manipulate the data.

The Perl code is contained in an `EXPRBODY` element. The Zinc Perl module provides classes and methods for gaining access to and manipulating the current record. Objects accessed from Perl are copies of the current data; changing the Perl objects has no effect on the current data until a function is called to explicitly copy the Perl objects back.

Endeca recommends that you perform record manipulation with the PERL_MANIPULATOR element rather than with the EXPRESSION and RECORD_MANIPULATOR elements. However, if you need to access and modify dimension sources (such as a dimension adapter or dimension server) you should still use expressions such as DVAL PERL. The Perl manipulator does not access dimension sources.

See the EXPRESSION element for DTD and attribute information.

Example

```
<EXPRESSION TYPE="INTEGER"  NAME="PERL">
   <EXPRBODY>
   ...your Perl code here. The VOID PERL topic has an example...
   </EXPRBODY>
</EXPRESSION>
```

INTEGER PROP_EXISTS

INTEGER expressions return integers (whole numbers). INTEGER expressions can be used to combine expressions, do arithmetic, and test conditions for conditional evaluation. The INTEGER PROP_EXISTS expression checks for a specified property on each record being processed.

The name of the property is specified in a PROP_NAME expression node. The expression returns the number of values of the property on each record. For example, if a record has three values from the “Color” property, the PROP_EXISTS expression would return 3. If the record has no values for the “Color” property, it would return 0. INTEGER PROP_EXISTS is useful as the condition expression in a VOID IF expression.

See the EXPRESSION element for DTD and attribute information.

Example

As part of an INTEGER AND expression, this example uses two INTEGER PROP_EXISTS sub-expressions to test whether the specified PROP_NAME exists.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="AND">
      <EXPRESSION TYPE="INTEGER" NAME="MATH">
         <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
         <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
         <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
            <EXPRNODE NAME="PROP_NAME" VALUE="CATEGORY_ID"/>
         </EXPRESSION>
         <EXPRESSION TYPE="INTEGER" NAME="CONST">
            <EXPRNODE NAME="VALUE" VALUE="0"/>
         </EXPRESSION>
      </EXPRESSION>
      <EXPRESSION TYPE="INTEGER" NAME="MATH">
         <EXPRNODE NAME="TYPE" VALUE="INTEGER"/>
         <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
         <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
            <EXPRNODE NAME="PROP_NAME" VALUE="SALESRANK"/>
         </EXPRESSION>
         <EXPRESSION TYPE="INTEGER" NAME="CONST">
            <EXPRNODE NAME="VALUE" VALUE="0"/>
         </EXPRESSION>
      </EXPRESSION>
   </EXPRESSION>
```

```
   <EXPRESSION TYPE="VOID" NAME="REMOVE_RECORD"/>
</EXPRESSION>
```

PROPERTY expressions

PROPERTY ALL

PROPERTY expressions return properties (name-value pairs). They are typically used to provide data to other expressions. The PROPERTY ALL expression returns all the values from all the properties on the current record.

There are no EXPRNODE elements to configure it.

See the EXPRESSION element for DTD and attribute information.

Example

```
<EXPRESSION TYPE="PROPERTY" NAME="ALL"/>
```

PROPERTY DVAL

PROPERTY expressions return properties (name-value pairs). They are typically used to provide data to other expressions. The PROPERTY DVAL expression creates a property value for each dimension value the current record has from the specified dimension.

A PROP_NAME expression node specifies the name of the property to be created. The dimension can be specified using either a DIMENSION_ID or a DIMENSION_NAME expression node. By default, values for the property are created containing the name of each dimension value assigned to the record from the specified dimension. If the FULL_PATH expression node is specified with a value of TRUE, then the names of all dimension values in the path from the dimension root to the assigned dimension value are concatenated (separated by '/') and used instead of the dimension value name.

See the EXPRESSION element for DTD and attribute information.

Example

This example creates the Price property for the specified dimension value in the record being processed.

```
<EXPRESSION TYPE="VOID" NAME="ADD_PROP">
   <EXPRESSION TYPE="PROPERTY" NAME="DVAL">
      <EXPRNODE NAME="DIMENSION_ID" VALUE="300"/>
      <EXPRNODE NAME="PROP_NAME" VALUE="Price"/>
      <EXPRNODE NAME="FULL_PATH" VALUE="TRUE"/>
   </EXPRESSION>
</EXPRESSION>
```

PROPERTY IDENTITY

PROPERTY expressions return properties (name-value pairs). They are typically used to provide data to other expressions. The PROPERTY IDENTITY expression returns the specified property from the current record.

The name of the property to return is specified in a PROP_NAME expression node.

See the EXPRESSION element for DTD and attribute information.

Example

This example assigns the dimension value 2090 to records with the property name FORMAT. The property name is specified within the PROPERTY IDENTITY sub-expression.

```
<EXPRESSION TYPE="VOID" NAME="ADD_DVAL">
   <EXPRESSION TYPE="DVAL" NAME="MATCH">
      <EXPRNODE NAME="DIMENSION_ID" VALUE="2090"/>
         <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
            <EXPRNODE NAME="PROP_NAME" VALUE="FORMAT"/>
         <EXPRNODE NAME="LOG" VALUE="salog"/>
         </EXPRESSION>
    </EXPRESSION>
</EXPRESSION>
```

PROPERTY NVL

PROPERTY expressions return properties (name-value pairs). They are typically used to provide data to other expressions. The PROPERTY NVL expression evaluates its sub-expressions and returns the properties from the first sub-expression that has a non-empty result.

This is useful if there are several different properties that a record may or may not have, but only one of them should be used in an expression. For example, a record may have a property for a home address or for a business address. PROPERTY NVL expression evaluates each possibility and returns the first address it locates.

Note that a sub-expression that returns empty strings is returning values (empty values). For example, a record may not have a value for the “Color” property; however, if the FILTER_EMPTY_PROPS attribute of the RECORD_ADAPTER is not set to true, an identity expression referencing the property "Color” still has a return value (that is, the property name “Color” with an empty value).

An NVL expression may therefore return properties with empty values; it skips only properties that do not exist. There are no EXPRNODE elements to configure PROPERTY NVL.

See the EXPRESSION element for DTD and attribute information.

Example

As suggested in the above example, this expression returns the first address property that Forge locates in the record.

```
<EXPRESSION TYPE="PROPERTY" NAME="NVL">
   <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
      <EXPRNODE NAME="PROP_NAME" VALUE="address_home"/>
   </EXPRESSION>
   <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
      <EXPRNODE NAME="PROP_NAME" VALUE="address_work"/>
   </EXPRESSION>
</EXPRESSION>
```

PROPERTY PERL

PROPERTY expressions return properties (name-value pairs). They are typically used to provide data to other expressions. The PROPERTY PERL expression uses Perl to manipulate the data.

The Perl code is contained in an EXPRBODY element. The Zinc Perl module provides classes and methods for gaining access to and manipulating the current record. Objects accessed from Perl are copies of the current data; changing the Perl objects has no effect on the current data until a function is called to explicitly copy the Perl objects back.

Endeca recommends that you perform record manipulation with the PERL_MANIPULATOR element rather than with the EXPRESSION and RECORD_MANIPULATOR elements. However, if you need to access and modify dimension sources (such as a dimension adapter or dimension server) you should still use expressions such as DVAL PERL. The Perl manipulator does not access dimension sources.

See the EXPRESSION element for DTD and attribute information.

Example

```
<EXPRESSION TYPE="PROPERTY"  NAME="PERL">
   <EXPRBODY>
   ...your Perl code here. The VOID PERL topic has an example...
   </EXPRBODY>
</EXPRESSION>
```

VOID expressions

VOID ADD_DVAL

VOID expressions return no value but are used to perform other work. The VOID ADD_DVAL expression adds dimension values to the current record.

The dimension values to add are given by one or more sub-expressions of type DVAL.

See the EXPRESSION element for DTD and attribute information.

Example

This example adds the dimension value to the record being processed.

```
<EXPRESSION TYPE="VOID" NAME="ADD_DVAL">
   <EXPRESSION TYPE="DVAL" NAME="CONST">
      <EXPRNODE NAME="DIMENSION_ID" VALUE="2090"/>
      <EXPRNODE NAME="DVAL_ID" VALUE="2091"/>
   </EXPRESSION>
</EXPRESSION>
```

Related Links

VOID expressions on page 23

VOID ADD_DVAL_PROP

VOID expressions return no value but are used to perform other work. The VOID ADD_DVAL_PROP expression adds dimension value attributes to a dimension value.

This expression is unique in that it does not modify the current record. The attributes are information for an Endeca application's user interface; the attributes are not record processing information for Forge or the MDEX Engine. For example, you might use attributes to indicate the display color or location of a dimension value. See the *Endeca Basic Development Guide* for details.

The dimension to modify can be given in either a DIMENSION_NAME or a DIMENSION_ID expression node; the dimension value to modify can be given in either a DVAL_PATH or DVAL_ID expression node. The properties to be added are given by one or more PROPERTY sub-expressions.

By default, if the dimension value already has attributes attached, the new attributes are simply added. If the optional REPLACE expression node is set to TRUE, any duplicate attributes are removed prior to adding the new ones.

By default, the modifications are made to the dimension value immediately. In some cases (for example with AutoGen), the specified dimension value may not exist at the time the expression is evaluated, so Forge should wait and make the specified changes after all records have been processed. To do this, the optional DELAY expression node should be set to TRUE.

See the EXPRESSION element for DTD and attribute information.

Example

This expression adds an attribute named display to the Red dimension value.

```
<EXPRESSION TYPE="VOID" NAME="ADD_DVAL_PROP">
   <EXPRNODE NAME="DIMENSION_NAME" VALUE="Colors"/>
   <EXPRNODE NAME="DVAL_PATH" VALUE="Colors/Red"/>
   <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
      <EXPRNODE NAME="PROP_NAME" VALUE="display"/>
   </EXPRESSION>
</EXPRESSION>
```

VOID CLEAN_DVALS

VOID expressions return no value but are used to perform other work. The VOID CLEAN_DVALS expression removes ancestor classifications from records.

CLEAN_DVALS requires one or more DIMENSION_ID expression nodes to tell it which dimensions to clean. You can also specify dimensions from more than one DIMENSION_SOURCE if necessary. One DIMENSION_SOURCE is required.

If a dimension value and one or more of its ancestors are assigned to a record, a CLEAN_DVALS expression deletes the ancestor dimension values, leaving only the child dimension value on the record. A dimension value and one or more of its ancestors would be assigned to a record in a situation where an ancestor and dimension leaf value are both properties of a single record. This is the case in the following example for the properties “Blue” and “Sky Blue”.

For example, suppose one navigation path within a “Colors” dimension looked like this: Colors->Blue->Sky Blue. If a record has a property value of “Blue” and a property value of “Sky Blue”, then both the parent dimension value “Blue” and its child “Sky Blue” will be assigned to it. A CLEAN_DVALS expression would remove the dimension value “Blue” from the record. If there were more levels of hierarchy (for example, if the dimension value “Blue” were a grandparent or great-grandparent), the CLEAN_DVALS expression would work in the same way; only the child dimension value “Sky Blue” would remain on the record.

See the EXPRESSION element for DTD and attribute information.

Example

This example cleans ancestor dimension values from the indicated dimension.

```
<EXPRESSION TYPE="VOID" NAME="CLEAN_DVALS">
   <EXPRNODE NAME="DIMENSION_ID" VALUE="70000" />
```

```
   <EXPRNODE NAME="DIMENSION_SOURCE" VALUE="DimensionServer" />
</EXPRESSION>
```

VOID CONVERTTOTEXT

VOID expressions return no value but are used to perform other work. The VOID CONVERTTOTEXT expression extracts document content, converts it to text, and assigns the text to a record.

This expression is available as part of the optional Document Conversion Module. Recall that the RE¬ TRIEVE_URL expression fetches a document's content and writes the content to a file. The Ende¬ ca.Document.Body property stores the absolute path of the file that contains the document's content. CONVERTTOTEXT read the path and converts the content of the indicated file to text. The text is then assigned to a record as a property with the name Endeca.Document.Text. If the expression fails, a warning is logged, and the property is not assigned to the record.

The following optional expression nodes modify the behavior of VOID CONVERTTOTEXT:

- TIMEOUT - Specifies the maximum time allowed to convert a document. The default value is 300 seconds.
- RESPONSE_TIMEOUT - Specifies the messaging time out between Forge and the converter process. The default value is 30 seconds.
- CONVERT_EMBEDDED - If set to TRUE, specifies that embedded documents will also be extracted and converted. If this option is not used, the default is FALSE.

See the EXPRESSION element for DTD and attribute information.

Example

This example converts Endeca.Document.Body to text if the property exists.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="PROP_EXISTS">
      <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Body"/>
   </EXPRESSION>
   <EXPRESSION TYPE="VOID" NAME="CONVERTTOTEXT"/>
</EXPRESSION>
```

This example shows how to use the CONVERT_EMBEDDED option to process embedded documents.

```
<EXPRESSION TYPE="VOID" NAME="CONVERTTOTEXT">
   <EXPRNODE NAME="CONVERT_EMBEDDED" VALUE="TRUE" />
</EXPRESSION>
```

VOID CREATE

VOID expressions return no value but are used to perform other work. The VOID CREATE expression creates a new property on the current record.

The name of the property is specified either by a PROP_NAME expression node, or by the first sub-expression, which may be of type STRING, PROPERTY, INTEGER, or FLOAT. The value for the property comes from a second sub-expression of type STRING, PROPERTY, INTEGER, or FLOAT.

See the EXPRESSION element for DTD and attribute information.

Example

This example creates a new property called `Document-Digest`.

```
<EXPRESSION TYPE="VOID" NAME="CREATE">
   <EXPRNODE NAME="PROP_NAME" VALUE="Document-Digest"/>
   <EXPRESSION TYPE="STRING" NAME="DIGEST">
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Body"/>
      </EXPRESSION>
   </EXPRESSION>
</EXPRESSION>
```

VOID EXPORT_PROP

`VOID` expressions return no value but are used to perform other work. The `VOID EXPORT_PROP` expression writes a given property to a file and replaces the property value with the value of the newly created file's URL.

Writing a property value to a file is useful when the property is a long text description. You can save memory by writing the property to a file and accessing the file only when necessary via the new URL property value. You can use `VOID REMOVE_EXPORTED_PROP` to delete this file and the property pointing to the file.

The property to export can be specified in either one of the following ways:

- Use the `PROP_NAME` expression node to specify the name of the property to export.
- Use a `STRING` expression to generate the name of the file to export. This mode is useful when crawling files: the file name is generated using a `STRING DIGEST` of the `Endeca.Identifier` property and the expression generates a different file name for each URL identifier.

The following expression nodes can modify `EXPORT_PROP`:

- `URL` - Specifies the base URL that files are written to. This value may be either an absolute path or a path relative to the location of `Pipeline.epx`.
- `PREFIX` - Specifies a file name prefix to use when Forge writes the property value to a file.

See the `EXPRESSION` element for DTD and attribute information.

Example

This example exports properties Prop1 and Prop2 from the props directory.

```
<EXPRESSION TYPE="VOID" NAME="EXPORT_PROP">
   <EXPRNODE NAME="PROP_NAME" VALUE="Prop1"/>
   <EXPRNODE NAME="PROP_NAME" VALUE="Prop2"/>
   <EXPRNODE NAME="URL" VALUE="props"/>
   <EXPRNODE NAME="PREFIX" VALUE="out."/>
</EXPRESSION>
```

VOID ID_LANGUAGE

`VOID` expressions return no value but are used to perform other work. The `VOID ID_LANGUAGE` expression identifies the language of a specified property and then adds a language identifier property to a record.

By default, the language identifier property is named `Endeca.Document.Language`. The value of the property is the ISO 639 code that corresponds to the language. For example, for Japanese the

value of Endeca.Document.Language would be JA. For English, the value of Endeca.Document.Language would be EN.

The VOID ID_LANGUAGE expression can identify 118 language and encoding pairs as described in the Language Support Table. See the RECORD_ADAPTER and its ENCODING attribute for more information about language encoding support for source data.

This expression is particularly useful if you want to use language types to control linguistic features of your Endeca-enabled application. For example, Forge can tag records with the identified language, and you can present language as a navigation parameter in an Endeca-enabled application. Similarly, language identification can be used to only show, for example, Italian documents to a user in an Italian user interface.

If you have a SPIDER set up to extract the body of a document, whose contents are stored by default as a property value of Endeca.Document.Text, you can then run ID_LANGUAGE against Endeca.Document.Text to identify the language of the document contents.

The following expression nodes modify ID_LANGUAGE:

- PROPERTY - Specifies the name of the property on which to perform language identification.
- LANG_PROP_NAME - Specifies the name of the property to store the language. The default value is Endeca.Document.Language.
- LANG_ID_BYTES - Specifies the number of bytes Forge uses to determine the language. A larger number provides a more accurate determination, but requires more processing time. The default value is 300 bytes.

See the EXPRESSION element for DTD and attribute information.

Example

This example exports properties Prop1 and Prop2 from the props directory.

```
<EXPRESSION TYPE="VOID" NAME="ID_LANGUAGE">
   <EXPRNODE NAME="PROPERTY" VALUE="Description"/>
   <EXPRNODE NAME="LANG_PROP_NAME" VALUE="Endeca.Document.Language"/>
</EXPRESSION>
```

VOID IF

VOID expressions return no value but are used to perform other work. The VOID IF expression provides a way to perform conditional evaluation.

The sub-expressions are grouped into clauses: the first clause consists of all the sub-expressions up to the first EXPRNODE element (if any) and any subsequent clauses consist of the sub-expressions between EXPRNODE elements. The first clause is an IF clause; the first sub-expression is a condition, and must be of type INTEGER. Subsequent sub-expressions form the action, and must be of type VOID. If the condition evaluates to anything other than zero, then all of the actions are evaluated, in order. If the condition evaluates to zero, then processing moves to the next clause.

A clause introduced by an ELSE_IF expression node (EXPRNODE) behaves just like the initial IF clause. ELSE_IF clauses are optional. If included, there may be any number. For a sample usage, see the second example below. In a clause introduced by an ELSE expression node, all sub-expressions form an action, and must be of type VOID. The actions are evaluated in order. The ELSE clause is optional. If included, it must come last. There can be at most one ELSE clause.

See the EXPRESSION element for DTD and attribute information.

Example 1

This example evaluates whether the Region property is equal to the constant Other Italy. If two are equal, then the REMOVE expression deletes the property.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="MATH">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="Region"/>
       </EXPRESSION>
      <EXPRESSION TYPE="STRING" NAME="CONST">
         <EXPRNODE NAME="VALUE" VALUE="Other Italy"/>
      </EXPRESSION>
   </EXPRESSION>
   <EXPRESSION TYPE="VOID" NAME="REMOVE">
      <EXPRNODE NAME="PROP_NAME" VALUE="Region"/>
   </EXPRESSION>
</EXPRESSION>
```

Example 2

This example implements the following logic using ELSE and ELSE_IF expression nodes:

```
   if (Endeca.Title == "Ad Rotator Test")
      Create Property AdRotate with Property Value "Yes"
   else if (Endeca.Title == "Sample Pages")
      Create Property "An Index" with Property Value "Yes"
   else
      Create Property "NoMatch" with Property Value "Nothing"
   end if
```

The example is as follows:

```
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">

  <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
    <EXPRNODE NAME="TYPE" VALUE="STRING"/>
    <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
    <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
      <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Title"/>
    </EXPRESSION>
    <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
      <EXPRNODE NAME="VALUE" VALUE="Ad Rotator Test"/>
    </EXPRESSION>
  </EXPRESSION>

  <EXPRESSION LABEL="" NAME="CREATE" TYPE="VOID" URL="">
    <EXPRNODE NAME="PROP_NAME" VALUE="AdRotate"/>
    <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
      <EXPRNODE NAME="VALUE" VALUE="Yes"/>
    </EXPRESSION>
  </EXPRESSION>

  <EXPRNODE NAME="ELSE_IF" VALUE=""/>

  <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
    <EXPRNODE NAME="TYPE" VALUE="STRING"/>
    <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
    <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
      <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Title"/>
```

```
    </EXPRESSION>
    <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
      <EXPRNODE NAME="VALUE" VALUE="Sample Pages"/>
    </EXPRESSION>
  </EXPRESSION>

  <EXPRESSION LABEL="" NAME="CREATE" TYPE="VOID" URL="">
    <EXPRNODE NAME="PROP_NAME" VALUE="An Index"/>
    <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
      <EXPRNODE NAME="VALUE" VALUE="Yes"/>
    </EXPRESSION>
  </EXPRESSION>

  <EXPRNODE NAME="ELSE" VALUE=""/>

  <EXPRESSION LABEL="" NAME="CREATE" TYPE="VOID" URL="">
    <EXPRNODE NAME="PROP_NAME" VALUE="NoMatch"/>
    <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
      <EXPRNODE NAME="VALUE" VALUE="Nothing"/>
    </EXPRESSION>
  </EXPRESSION>

</EXPRESSION>
```

VOID IMPORT_PROP

`VOID` expressions return no value but are used to perform other work. The `VOID IMPORT_PROP` expression imports a property value from a specified file.

Typically, files containing property values were created using the complementary expression `VOID EXPORT_PROP`. Another typical use of `IMPORT_PROP` is to read a document body created by `VOID RETRIEVE_URL`. (This is useful for HTML pre-processing with a spider.)

The property to import can be specified in either one of the following ways:

- Use the `PROP_NAME` expression node to specify the name of the property to import. The current value of the property is the file name. The current value is replaced with the value read from the file. All properties with a given name are affected in this mode.
- Use a `STRING` expression to generate the name of the file to import. Typically, the file name is generated using a `STRING DIGEST` of the Endeca.Identifier property. If this expression is present, then one property can be imported per record. This expression generates a different file name for each record. Existing values are untouched.

The following expression nodes modify `IMPORT_PROP`. Several of these nodes are the same as those used to identify the property during export in `EXPORT_PROP`.

- `URL` - Specifies the URL that files are imported from. This value may be either an absolute path or a path relative to the location of `Pipeline.epx`.
- `PREFIX` - Specifies any prefix used in the file name to remove. This value often corresponds to the value of `PREFIX` in the `VOID EXPORT_PROP` expression.
- `REMOVE_FILES` - Specifies whether to delete files, after importing their property values. When set to `TRUE`, the files are deleted. The default value is `FALSE`.
- `ENCODING` - Specifies the encoding that should be used during import. If desired, this encoding may be overriden by `ENCODING_PROP`.
- `ENCODING_PROP` - Specifies the name of the property containing the encoding. The `RETRIEVE_URL` expression creates this property with a default property name of `Endeca.Document.Encoding`.

- ENCODING_ID_BYTES - Specifies the number of bytes used to identify the encoding. This value defaults to its maximum of 1 MB and can be reduced if necessary to optimize performance.

See the EXPRESSION element for DTD and attribute information.

Example

This example imports properties Prop1 and Prop2 from the props directory. After importing, the expression deletes the files.

```
<EXPRESSION TYPE="VOID" NAME="IMPORT_PROP">
   <EXPRNODE NAME="PROP_NAME" VALUE="Prop1"/>
   <EXPRNODE NAME="PROP_NAME" VALUE="Prop2"/>
   <EXPRNODE NAME="URL" VALUE="props"/>
   <EXPRNODE NAME="PREFIX" VALUE="out."/>
   <EXPRNODE NAME="REMOVE_FILES" VALUE="TRUE"/>
</EXPRESSION>
```

VOID PARSE_DOC

VOID expressions return no value but are used to perform other work. The VOID PARSE_DOC expression obtains metadata and extracts text from documents and adds the metadata and document text in the form of property values to a record.

Both text/plain and text/html files can be extracted from documents by this expression; other file types are passed to the Document Conversion Module converters for parsing. See "Implementing the Endeca Crawler" in the *Endeca Forge Guide* for a description of each generated property that PARSE_DOC adds to the record.

The following list describes the optional expression nodes that can modify PARSE_DOC:

- FILE_PATH - Specifies whether the expression interprets the property value as a file path (to the contents of the file) or the contents of the file itself. TRUE interprets the property value as a file path. FALSE interprets the property value as the contents of the file.
- PARSE_META - Indicates whether to extract metadata of a document. TRUE extracts metadata; FALSE does not. The default value is TRUE.
- PARSE_TEXT - Indicates whether to extract the body text of a document. TRUE extracts text; FALSE does not. The default value is TRUE.
- MIMETYPE_PROP - Describes the name of the property containing the content type. The RE¬TRIEVE_URL expression creates this property with a default property name of Endeca.Docu¬ment.MimeType. You do not need to modify this name unless desired.
- ENCODING_PROP - Describes the name of the property containing the encoding. The RE¬TRIEVE_URL expression creates this property with a default property name of Endeca.Docu¬ment.Encoding. You do not need to modify this name unless desired.
- BODY_PROP - Describes the name of the property containing the document body. The RE¬TRIEVE_URL expression creates this property with a default property name of Endeca.Docu¬ment.Body. You do not need to modify this name unless desired.
- TEXT_PROP Describes the name of the property to put document text into. The RETRIEVE_URL expression creates this property with a default property name of Endeca.Document.Text. You do not need to modify this name unless desired.

See the EXPRESSION element for DTD and attribute information.

Example

This example parses the property and adds the contents to the record being processed.

```
<EXPRESSION TYPE="VOID" NAME="PARSE_DOC">
   <EXPRNODE NAME="BODY_PROP" VALUE="Endeca.Document.Body"/>
   <EXPRNODE NAME="FILE_PATH" VALUE="TRUE"/>
</EXPRESSION>
```

VOID PERL

VOID expressions return no value but are used to perform other work. The VOID PERL expression uses Perl to manipulate the data.

The Perl code is contained in an EXPRBODY element. The Zinc Perl module provides classes and methods for gaining access to and manipulating the current record.

Objects accessed from Perl are copies of the current data; changing the Perl objects has no effect on the current data until a function is called to explicitly copy the Perl objects back.

Endeca recommends that you perform record manipulation with the PERL_MANIPULATOR element rather than with the EXPRESSION and RECORD_MANIPULATOR elements. However, if you need to access and modify dimension sources (such as a dimension adapter or dimension server) you should still use expressions such as DVAL PERL. The Perl manipulator does not access dimension sources.

See the EXPRESSION element for DTD and attribute information.

Example

```
<EXPRESSION TYPE=”PROPERTY”  NAME=”PERL”>
   <EXPRBODY>
   <![CDATA[
   ### IF ITEM HAS "LPRINT" PROPERTY WITH VALUE OF "Y",
   ### REMOVE CURRENT "FORMAT" PROPERTY,
   ### AND CREATE NEW ONE WITH VALUE OF "LP"

   my @lprint = get_props_by_name("LPRINT");

   my @format = get_props_by_name("FORMAT");

   if (-1 != $#lprint) {
      my $large = ($lprint[0])->value();
      if ($large =~ /Y/) {
         if (-1 != $#format) {
            remove_props("FORMAT");
         }

      my $new_prop = new Zinc::PropVal("FORMAT", "LP");

      add_props($new_prop);
      }
   }

   ]]>
   </EXPRBODY>
</EXPRESSION>
```

VOID REMOVE

VOID expressions return no value but are used to perform other work. The VOID REMOVE expression removes the specified properties from the current record.

The names of the properties to be removed are given in PROP_NAME expression nodes.

See the EXPRESSION element for DTD and attribute information.

Example

This example evaluates whether the Region property is equal to the constant Other Italy. If two are equal, then the REMOVE expression deletes the property.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="MATH">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="Region"/>
      </EXPRESSION>
      <EXPRESSION TYPE="STRING" NAME="CONST">
         <EXPRNODE NAME="VALUE" VALUE="Other Italy"/>
      </EXPRESSION>
   </EXPRESSION>
   <EXPRESSION TYPE="VOID" NAME="REMOVE">
      <EXPRNODE NAME="PROP_NAME" VALUE="Region"/>
   </EXPRESSION>
</EXPRESSION>
```

VOID REMOVE_EXPORTED_PROP

VOID expressions return no value but are used to perform other work. The VOID REMOVE_EXPORT¬ED_PROP expression deletes the file containing the value of an exported property and also the property value itself, if desired.

Use REMOVE_EXPORTED_PROP to remove a file created by the VOID EXPORT_PROP or VOID RE¬TRIEVE_URL expressions. The following expression nodes can modify REMOVE_EXPORTED_PROP:

- PROP_NAME - Specifies the name of the property to remove.
- URL - Specifies the URL that files were written to. This value may be either an absolute path or a path relative to the location of Pipeline.epx.
- PREFIX - Specifies any prefix used in the file name to remove. This value often corresponds to the value of PREFIX in the VOID EXPORT_PROP expression.
- REMOVE_PROPS - Specifies whether to remove the property from the record after deleting the file where the property was stored. TRUE removes the property from the record after removing the corresponding file. FALSE does not remove the property.

See the EXPRESSION element for DTD and attribute information.

Example

As the COMMENT element indicates, this example removes the temporary file created by EXPORT_PROP.

```
<EXPRESSION TYPE="VOID" NAME="REMOVE_EXPORTED_PROP">
<COMMENT>This expression removes the temporary file that is created
on disk by the RETRIEVE_URL expression.</COMMENT>
   <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Body"/>
```

```
   <EXPRNODE NAME="REMOVE_PROPS" VALUE="TRUE"/>
</EXPRESSION>
```

VOID REMOVE_RECORD

VOID expressions return no value but are used to perform other work. The VOID REMOVE_RECORD expression removes the current record.

Processing of the record stops, and the next record is retrieved. REMOVE_RECORD is typically used within an IF expression to remove records that meet or do not meet certain criteria.

See the EXPRESSION element for DTD and attribute information.

Example

This example removes the record if the value of Name and Text are the same.

```
<EXPRESSION TYPE="VOID" NAME="IF">
   <EXPRESSION TYPE="INTEGER" NAME="MATH">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="Name"/>
      </EXPRESSION>
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="Text"/>
      </EXPRESSION>
   </EXPRESSION>
   <EXPRESSION TYPE="VOID" NAME="REMOVE_RECORD"/>
</EXPRESSION>
```

VOID RENAME

VOID expressions return no value but are used to perform other work. The VOID RENAME expression changes the name of the specified property for the current record.

The name of the property to change is given in an OLD_NAME expression node, and the new name is given in a NEW_NAME expression node. Only the property name is affected—the property values stay the same.

See the EXPRESSION element for DTD and attribute information.

Example

This example renames PropPrice to Price.

```
<EXPRESSION TYPE="VOID" NAME="RENAME">
   <EXPRNODE NAME="OLD_NAME" VALUE="PropPrice"/>
   <EXPRNODE NAME="NEW_NAME" VALUE="Price"/>
</EXPRESSION>
```

VOID RETRIEVE_URL

VOID expressions return no value but are used to perform other work. The VOID RETRIEVE_URL expression processes records that have a URL property by retrieving the URL, its corresponding document content, and metadata.

RETRIEVE_URL requires a STRING sub-expression that names a file created to store the document content from the URL. The STRING DIGEST expression is typically used to generate the file.

Forge adds the location of the file, the document content, and other values to the record as property values. The file containing the document content must be unique for each record or Forge overwrites the content when processing subsequent records.

Parameters that affect how this expression retrieves URLs can be expressed as record properties to configure URL retrieval. These parameters include connection time outs (Endeca.Fetch.Connect¬Timeout), data transfer rates (Endeca.Fetch.TransferRateLowSpeedLimit), the use of proxy servers (Endeca.Fetch.Proxy), and so on. See "Implementing the Endeca Crawler" in the *Endeca Forge Guide* for information about metadata properties and configuration properties that the expression retrieves or stores with the record.

The following optional expression nodes modify the behavior of VOID RETRIEVE_URL:

- BODY_PROP_NAME - Specifies the name of the property containing the document body. The default value of this property is Endeca.Document.Body.
- URL_PROP_NAME - Specifies the name of the property that contains the URL to retrieve. Only one URL is retrieved per record. The default value of this property is Endeca.Identifier.
- REVISION_PROP_NAME - Specifies the name of the property that contains the URL's revision information. The default value of this property is Endeca.Document.Revision.
- KEY_RING - Specifies the path to a Key_ring.xml file that contains the authentication information which a SPIDER uses when communicating with a host computer. Specify the path to this file in the VALUE attribute. The path to the file may be absolute or relative to the location of the Pipeline.epx file.

See the EXPRESSION element for DTD and attribute information.

Example

This example generates a file name for the retrieved file and it specifies that a Key_ring.xml should be used for authentication.

```
<EXPRESSION TYPE="VOID" NAME="RETRIEVE_URL">
   <!-- this expression generates a filename for the retrieved file -->
   <EXPRESSION TYPE="STRING" NAME="CONCAT">
      <EXPRESSION TYPE="STRING" NAME="CONST">
      <EXPRNODE NAME="VALUE" VALUE="&cwd;"/>
      </EXPRESSION>
      <EXPRESSION TYPE="STRING" NAME="DIGEST">
         <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
            <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Identifier"/>
         </EXPRESSION>
      </EXPRESSION>
      </EXPRESSION>
   <!-- this expression node specifies the path to the key ring file -->
   <EXPRNODE NAME="KEY_RING" VALUE="key_ring.xml"/>
</EXPRESSION>
```

VOID SPLIT

VOID expressions return no value but are used to perform other work. The VOID SPLIT expression splits the values of a single property into multiple values of a new property, or into multiple properties.

Configure the expression as follows:

- Specify the property to split in an OLD_NAME expression node.

- Specify the property to contain the new values in a NEW_NAME expression node.
- Specify the delimiter to split on in a SPLIT expression node.

After performing the split, Forge trims leading and trailing white space from the new property values, so leading or trailing spaces do not have to be included in the delimiter. For example, if the value of the “Colors” property for a certain record is “red, blue, green”, and you split that value on the comma delimiter, with a new name of “Hue”, the output is three separate properties: “Hue1”=“red”, “Hue2”=“blue”, “Hue3”=“green”.

The default value of the optional ENUMERATE expression node is TRUE. If ENUMERATE is set to FALSE, all of the new values are assigned to a single new property. In the previous example, the result would be a single “Hue” property with the values “red”, “blue”, and “green,'' instead of three separate properties.

See the EXPRESSION element for DTD and attribute information.

Example

The example described above is expressed as follows:

```
<EXPRESSION TYPE="VOID" NAME="SPLIT">
   <EXPRNODE NAME="OLD_NAME" VALUE="Colors"/>
   <EXPRNODE NAME="NEW_NAME" VALUE="Hue"/>
   <EXPRNODE NAME="SPLIT" VALUE=","/>
   <EXPRNODE NAME="ENUMERATE" VALUE="TRUE"/>
</EXPRESSION>
```

VOID STRATIFY

VOID expressions return no value but are used to perform other work. The VOID STRATIFY expression identifies a Stratify Classification Server that classifies Endeca records.

For each record that passes through the record manipulator, the STRATIFY expression requests that the Stratify Classification Server classify a document indicated by Endeca.Document.Body. Forge sends the document as an attachment to a Stratify Classification Server. The Stratify Classification Server examines the document including the document’s structure and classifies it according to the classification model you developed in Stratify Taxonomy Manager. You indicate the classification model in the HIERARCHY_ID expression node. The Classification Server then sends back property values containing a Stratify topic name, a unique ID, and a confidence rating of the classification. Forge appends these values to the record for the document.

The following expression nodes are required in VOID STRATIFY:

- STRATIFY_HOST - Specifies the machine name or IP address of the Stratify Classification Server.
- STRATIFY_PORT - Specifies the port on which the Stratify Classification Server listens for requests from Forge.
- HIERARCHY_ID - Specifies the identifier of a Stratify classification model. To determine the VALUE of HIERARCHY_ID: First, navigate to the working directory of the Stratify Classification Server that contains your classification model and taxonomy files. This directory is typically located at <Stratify Install Directory>\ClassificationServer\ClassificationServer\ClassificationServerWorkDir\Taxonomy-N, where N is the number of the directory that contains the classification model you want to use with your Endeca project. (Your environment may have multiple \Taxonomy-N directories each containing different classification model and taxonomy files.) Second, note the number at the end of the of \Taxonomy-N directory. This number is the value of HIERARCHY_ID. For example, if the classification model you want to use is stored in ...\Taxonomy-2, then HIERARCHY_ID should have VALUE="2".

- IDENTIFIER_PROP_NAME - Specifies the unique ID for the Endeca record being processed. The default is Endeca.Identifier.
- BODY_PROP_NAME - Specifies the property that the Stratify Classification Server examines to classify the document. The default property is Endeca.Document.Body. You can provide either Endeca.Document.Body or Endeca.Document.Text. However, specifying Endeca.Document.Body provides better classification because Forge can send the document to Stratify Classification Server as an attachment, and Stratify Classification Server can use the attachment to determine structural information of the document that aids in classification. If you specify Endeca.Document.Text, Forge sends the converted text of the document without any of its structural information.

See the EXPRESSION element for DTD and attribute information.

Example

This example connects to the indicated Stratify Classification Server and requests that it classify the document indicated by Endeca.Document.Body using against hierarchy ID 1.

```
<EXPRESSION NAME="STRATIFY" TYPE="VOID" >
   <EXPRNODE NAME="STRATIFY_HOST" VALUE="10.0.0.999"/>
   <EXPRNODE NAME="STRATIFY_PORT" VALUE="7021"/>
   <EXPRNODE NAME="HIERARCHY_ID" VALUE="1"/>
   <EXPRNODE NAME="IDENTIFIER_PROP_NAME" VALUE="Endeca.Identifier"/>
<EXPRNODE NAME="BODY_PROP_NAME" VALUE="Endeca.Document.Body"/>
```

VOID UNIQUE

VOID expressions return no value but are used to perform other work. The VOID UNIQUE expression deletes every value of a property except the first.

The name of the property is given in a PROP_NAME expression node. For example, if the “Color” property has three values, “red”, “blue”, and “green”, then the UNIQUE expression removes the values “blue” and “green”, leaving just the value “red”.

See the EXPRESSION element for DTD and attribute information.

Example

The example described above is expressed as follows:

```
<EXPRESSION TYPE="VOID" NAME="UNIQUE">
   <EXPRNODE NAME="PROP_NAME" VALUE="Color"/>
</EXPRESSION>
```

VOID UPDATE

VOID expressions return no value but are used to perform other work. The VOID UPDATE expression changes the value of a property.

The name of the property to update is given in a PROP_NAME expression node, and the new value is given in a sub-expression of type INTEGER, FLOAT, STRING, or PROPERTY. If the property has multiple values, all values are changed.

See the EXPRESSION element for DTD and attribute information.

Example

This example updates records with the Endeca.Document.Body property by replacing "cwd" in paths with the actual current working directory.

```
<EXPRESSION TYPE="VOID" NAME="UPDATE">
   <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Body"/>
   <EXPRESSION TYPE="STRING" NAME="REPLACE">
      <EXPRNODE NAME="TARGET" VALUE="[cwd]"/>
      <EXPRNODE NAME="REPLACEMENT" VALUE="&cwd;"/>
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Body"/>
      </EXPRESSION>
   </EXPRESSION>
</EXPRESSION>
```

VOID UPDATE_RECORD

VOID expressions return no value but are used to perform other work. The VOID UPDATE_RECORD expression is used as part of a partial update pipeline.epx file to update existing records by adding, removing, or replacing dimensions, dimension values, or property values.

See the *Endeca Partial Updates Guide* for details about partial update processing. UPDATE_RECORD requires an ACTION expression node to indicate the type of update to perform. You can add additional PROP_ACTION, DIM_ACTION, and DVAL_ACTION expression nodes depending upon whether you want to modify properties, dimensions, or dimension values of the record. The following list further describes the expression nodes that refine the behavior of the UPDATE_RECORD expression:

- ACTION - Indicates the type of update to perform on a record. Valid values of the VALUE attribute are ADD, ADD_OR_REPLACE, DELETE, DELETE_OR_IGNORE, REPLACE, and UPDATE.
- PROP_ACTION - Modifies a property on the record. Valid values of the VALUE attribute are ADD, DELETE, REPLACE. A PROP_ACTION expression node must be followed by a PROP_NAME expression node that specifies the property to modify.
- DIM_ACTION - Modifies a dimension on the record. Valid values of the VALUE attribute are ADD, DELETE, REPLACE. A DIM_ACTION expression node must be followed by a DIMENSION_ID expression node that specifies the dimension ID of the dimension to modify.
- DVAL_ACTION - Modifies a dimension value on the record. The only valid value of the VALUE attribute is DELETE. A DVAL_ACTION expression node must be followed by a DVAL_ID expression node that specifies the dimension value ID of the dimension value to remove.

See the EXPRESSION element for DTD and attribute information.

Example

This example updates records in the Dgraph by replacing them with the values specified below.

```
<EXPRESSION TYPE="VOID" NAME="UPDATE_RECORD">
   <EXPRNODE NAME="ACTION" VALUE="UPDATE"/>
   <EXPRNODE NAME="PROP_ACTION" VALUE="REPLACE"/>
   <EXPRNODE NAME="PROP_NAME" VALUE="P_WineType1"/>
   <EXPRNODE NAME="PROP_ACTION" VALUE="REPLACE"/>
   <EXPRNODE NAME="PROP_NAME" VALUE="P_WineType2"/>
   <EXPRNODE NAME="DIM_ACTION" VALUE="REPLACE"/>
   <EXPRNODE NAME="DIMENSION_ID" VALUE="8000"/>
   <EXPRNODE NAME="PROP_ACTION" VALUE="REPLACE"/>
   <EXPRNODE NAME="PROP_NAME" VALUE="P_PriceStr"/>
</EXPRESSION>
```

STRING expressions

STRING CONCAT

STRING expressions return text strings. They are used to manipulate non-numeric data. The STRING CONCAT expression returns a string that is the concatenation of two or more values.

The values to be concatenated are given by sub-expressions, which can be of type STRING, INTEGER, FLOAT, or PROPERTY. There are no EXPRNODE elements to configure STRING CONCAT.

See the EXPRESSION element for DTD and attribute information.

Example

This example concatenates the value of the property ChapterNum and ChapterTitle with a space between them.

```
<EXPRESSION TYPE="VOID" NAME="CREATE">
   <EXPRNODE NAME="PROP_NAME" VALUE="ChapterNumTitle"/>
   <EXPRESSION TYPE="STRING" NAME="CONCAT">
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="ChapterNum"/>
      </EXPRESSION>
      <EXPRESSION TYPE="STRING" NAME="CONST">
         <EXPRNODE NAME="VALUE" VALUE=" "/>
      </EXPRESSION>
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="ChapterTitle"/>
      </EXPRESSION>
   </EXPRESSION>
</EXPRESSION>
```

STRING CONST

STRING expressions return text strings. They are used to manipulate non-numeric data. The STRING CONST expression returns the same string constant, specified in a VALUE expression node.

See the EXPRESSION element for DTD and attribute information.

Example

This example creates a property called Name by concatenating three values, one of which is a constant functioning as a term separator.

```
<EXPRESSION NAME="CREATE" TYPE="VOID">
   <EXPRNODE NAME="PROP_NAME" VALUE="Name"/>
   <EXPRESSION NAME="CONCAT" TYPE="STRING">
      <EXPRESSION NAME="IDENTITY" TYPE="PROPERTY">
         <EXPRNODE NAME="PROP_NAME" VALUE="file_name"/>
      </EXPRESSION>
      <EXPRESSION NAME="CONST" TYPE="STRING">
         <EXPRNODE NAME="VALUE" VALUE=", rev "/>
      </EXPRESSION>
      <EXPRESSION NAME="IDENTITY" TYPE="PROPERTY">
         <EXPRNODE NAME="PROP_NAME" VALUE="revision"/>
      </EXPRESSION>
```

```
   </EXPRESSION>
</EXPRESSION>
```

STRING DIGEST

STRING expressions return text strings. They are used to manipulate non-numeric data. The STRING DIGEST expression creates a property identifier that is a digest of a specified PROP_NAME expression node.

STRING DIGEST generates Message Digest 5 (MD5) digest strings, also called MD5 hashes and message digests. A STRING DIGEST expression requires a PROP_NAME expression node. Typically, a file name is generated using a STRING DIGEST of the Endeca.Identifier property.

See the EXPRESSION element for DTD and attribute information.

Examples

This example creates a digest identifier based on the Endeca.Identifier property.

```
<EXPRESSION TYPE="STRING" NAME="DIGEST">
   <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
      <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Identifier"/>
   </EXPRESSION>
</EXPRESSION>
```

This example shows how to use a STRING DIGEST expression as a sub-expression of the VOID RETRIEVE_URL expression, which is used to retrieve a document from its URL and store it in a file on disk.

```
<EXPRESSION LABEL="" NAME="RETRIEVE_URL" TYPE="VOID" URL="">
  <COMMENT>Retrieve the document and store it as a temporary
  file in the state directory, named with the digest (MD5 hash)
  of its URL.
  </COMMENT>
  <EXPRESSION LABEL="" NAME="CONCAT" TYPE="STRING" URL="">
    <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
     <EXPRNODE NAME="VALUE" VALUE="../partition0/state/"/>
    </EXPRESSION>
    <EXPRESSION TYPE="STRING" NAME="DIGEST">
     <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
      <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Identifier"/>
     </EXPRESSION>
    </EXPRESSION>
  </EXPRESSION>
</EXPRESSION>
```

STRING FORMAT

STRING expressions return text strings. They are used to manipulate non-numeric data. The STRING FORMAT expression returns the value from its sub-expression, converted to a floating-point number, and formatted as specified.

The sub-expression may be a PROPERTY, STRING or FLOAT expression. A PRECISION expression node sets the number of significant digits to include, and a SHOW_SIGN expression node sets whether or not to show the sign of the number.

See the EXPRESSION element for DTD and attribute information.

Example

This example takes a value 8.99, formats it with 3 significant digits, and returns 8.990.

```
<EXPRESSION TYPE="STRING" NAME="FORMAT">
   <EXPRNODE NAME="PRECISION" VALUE="3"/>
   <EXPRNODE NAME="SHOW_SIGN" VALUE="FALSE"/>
   <EXPRESSION TYPE="FLOAT" NAME="CONST">
      <EXPRNODE NAME="VALUE" VALUE="8.99"/>
   </EXPRESSION>
</EXPRESSION>
```

STRING PERL

STRING expressions return text strings. They are used to manipulate non-numeric data. The STRING PERL expression uses Perl to manipulate the data.

The Perl code is contained in an EXPRBODY element. The Zinc Perl module provides classes and methods for gaining access to and manipulating the current record. Objects accessed from Perl are copies of the current data; changing the Perl objects has no effect on the current data until a function is called to explicitly copy the Perl objects back.

Endeca recommends that you perform record manipulation with the PERL_MANIPULATOR element rather than with the EXPRESSION and RECORD_MANIPULATOR elements. However, if you need to access and modify dimension sources (such as a dimension adapter or dimension server) you should still use expressions such as DVAL PERL. The Perl manipulator does not access dimension sources.

See the EXPRESSION element for DTD and attribute information.

Example

```
<EXPRESSION TYPE="STRING" NAME="PERL">
   <EXPRBODY>
   ...your Perl code here. The VOID PERL topic has an example...
   </EXPRBODY>
</EXPRESSION>
```

STRING REPLACE

STRING expressions return text strings. They are used to manipulate non-numeric data. The STRING REPLACE expression returns a string where sections of the string have been replaced by another string.

The replacement occurs by taking an input string from a sub-expression, and replacing all occurrences of a sub-string specified by a TARGET expression node, with a replacement sub-string, specified by a REPLACEMENT expression node. The sub-expression may be a PROPERTY or a STRING expression.

See the EXPRESSION element for DTD and attribute information.

Example

This example replaces "cwd" in paths with the current working directory.

```
<EXPRESSION TYPE="VOID" NAME="UPDATE">
   <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Body"/>
      <EXPRESSION TYPE="STRING" NAME="REPLACE">
      <EXPRNODE NAME="TARGET" VALUE="[cwd]"/>
      <EXPRNODE NAME="REPLACEMENT" VALUE="&cwd;"/>
```

```
      <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
         <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Body"/>
      </EXPRESSION>
   </EXPRESSION>
</EXPRESSION>
```

Chapter 2

Data Foundry language support

Language Support Table

Using the `VOID ID_LANGUAGE` expression, Forge can identify the following 118 language and encoding pairs.

ARABIC CP1256 Microsoft Code Page 1256

ARABIC UTF-8 Unicode UTF-8

CATALAN ASCII ASCII

CATALAN ISO-8859-1 ISO-8859-1 (Latin 1)

CATALAN CP1252 Microsoft Code Page 1252

CATALAN UTF-8 Unicode UTF-8

CHINESE ASCII GB-Roman

CHINESE ASCII CNS-Roman

CHINESE GB GB2312-80

CHINESE CNS CNS 11643-1986

CHINESE EUC-CN EUC-CN

CHINESE BIG5 Big Five

CHINESE EUC DEC Hanzi Encoding

CHINESE BIG5-CP950 Microsoft Code Page 950

CHINESE Unicode Unicode UCS-2

CHINESE Unicode Unicode UTF-8

CZECH Latin2 ISO-8859-2 (Latin 2)

CZECH Latin2 Microsoft Code Page 1250

CZECH UTF-8 Unicode UTF-8

DANISH ASCII ASCII

DANISH ISO-8859-1 ISO-8859-1 (Latin 1)

DANISH CP1252 Microsoft Code Page 1252

DANISH UTF-8 Unicode UTF-8

DUTCH ASCII ASCII

DUTCH ISO-8859-1 ISO-8859-1 (Latin 1)

DUTCH CP1252 Microsoft Code Page 1252

DUTCH UTF-8 Unicode UTF-8

ENGLISH ASCII ASCII

ENGLISH ISO-8859-1 ISO-8859-1 (Latin 1)

ENGLISH CP1252 Microsoft Code Page 1252

ESTONIAN Latin4 ISO-8859-4 (Latin 4)

ESTONIAN Latin4 Microsoft Code Page 1257

ESTONIAN UTF-8 Unicode UTF-8

FINNISH ASCII ASCII

FINNISH ISO-8859-1 ISO-8859-1 (Latin 1)

FINNISH CP1252 Microsoft Code Page 1252

FINNISH UTF-8 Unicode UTF-8

FRENCH ASCII ASCII

FRENCH ISO-8859-1 ISO-8859-1 (Latin 1)

FRENCH CP1252 Microsoft Code Page 1252

FRENCH UTF-8 Unicode UTF-8

GERMAN ASCII ASCII

GERMAN ISO-8859-1 ISO-8859-1 (Latin 1)

GERMAN CP1252 Microsoft Code Page 1252

GERMAN UTF-8 Unicode UTF-8

GREEK Greek ISO-8859-7

GREEK Greek Microsoft Code Page 1253

GREEK UTF-8 Unicode UTF-8

HEBREW Hebrew ISO-8859-8

HEBREW Hebrew Microsoft Code Page 1255

HEBREW UTF-8 Unicode UTF-8

HUNGARIAN Latin2 ISO-8859-2 (Latin 2)

HUNGARIAN Latin2 Microsoft Code Page 1250

HUNGARIAN UTF-8 Unicode UTF-8

ICELANDIC ASCII ASCII

ICELANDIC ISO-8859-1 ISO-8859-1 (Latin 1)

ICELANDIC CP1252 Microsoft Code Page 1252

ICELANDIC UTF-8 Unicode UTF-8

ITALIAN ASCII ASCII

ITALIAN ISO-8859-1 ISO-8859-1 (Latin 1)

ITALIAN CP1252 Microsoft Code Page 1252

ITALIAN UTF-8 Unicode UTF-8

JAPANESE ASCII JIS-Roman

JAPANESE JIS ISO-2022-JP

JAPANESE JIS JIS X 0201-1976

JAPANESE JIS JIS X 0201-1997

JAPANESE JIS JIS X 0208-1983

JAPANESE JIS JIS X 0208-1990

JAPANESE JIS JIS X 0212-1983

JAPANESE JIS JIS X 0212-1990

JAPANESE EUC-JP EUC-JP

JAPANESE SJS Shift-JIS

JAPANESE JIS DEC Kanji

JAPANESE CP932 Microsoft Code Page 932

JAPANESE Unicode Unicode UCS-2

JAPANESE Unicode Unicode UTF-8

KOREAN ASCII KS-Roman

KOREAN KSC KS C 5861-1992

KOREAN KSC EUC-KR

KOREAN Unicode Unicode UCS-2

KOREAN Unicode Unicode UTF-8

LATVIAN Latin4 ISO-8859-4

LATVIAN Latin4 Microsoft Code Page 1257

LITHUANIAN Latin4 ISO-8859-4

LITHUANIAN Latin4 Microsoft Code Page 1257

LITHUANIAN UTF-8 Unicode UTF-8

NORWEGIAN ASCII ASCII

NORWEGIAN ISO-8859-1 ISO-8859-1 (Latin 1)

NORWEGIAN CP1252 Microsoft Code Page 1252

NORWEGIAN UTF-8 Unicode UTF-8

POLISH Latin2 ISO-8859-2 (Latin 2)

POLISH Latin2 Microsoft Code Page 1250

POLISH UTF-8 Unicode UTF-8

PORTUGUESE ASCII ASCII

PORTUGUESE ISO-8859-1 ISO-8859-1 (Latin 1)

PORTUGUESE CP1252 Microsoft Code Page 1252

PORTUGUESE UTF-8 Unicode UTF-8

ROMANIAN Latin2 ISO-8859-2 (Latin 2)

ROMANIAN Latin2 Microsoft Code Page 1250

ROMANIAN UTF-8 Unicode UTF-8

RUSSIAN ISO-8859-5 ISO-8859-5

RUSSIAN KOI8R KOI 8-R

RUSSIAN CP1251 Microsoft Code Page 1251

RUSSIAN UTF-8 Unicode UTF-8

SLOVAK Latin2 ISO-8859-2 (Latin 2)

SLOVAK UTF-8 Unicode UTF-8

SPANISH ASCII ASCII

SPANISH ISO-8859-1 ISO-8859-1 (Latin 1)

SPANISH CP1252 Microsoft Code Page 1252

SPANISH UTF-8 Unicode UTF-8

SWEDISH ASCII ASCII

SWEDISH ISO-8859-1 ISO-8859-1 (Latin 1)

SWEDISH CP1252 Microsoft Code Page 1252

SWEDISH UTF-8 Unicode UTF-8

THAI CP874 Microsoft Code Page 874

THAI UTF-8 Unicode UTF-8

TURKISH CP1254 Microsoft Code Page 1254

TURKISH UTF-8 Unicode UTF-8

Index